CHAPTER 10

Using ASP

Scripts have become the mainstay of many Web developers today for good reason: The learning curve is relatively short and the results are good. In fact, many Web-site developers use scripts for all parts of their development process simply because of the ease-of-development they provide. Chapters 8 and 9 should have demonstrated that ISAPI is a phenomenal technology, but that it requires a good knowledge of Visual C++ to use. In other words, ISAPI by itself probably isn't a total solution for your Web-server needs because it's overkill in many situations.

In this chapter, we'll begin by looking at the scripting language for IIS, Active Server Pages (ASP), from a Web programming perspective. We're not going to go into the basics of ASP. I'm assuming that you already have some basic knowledge of ASP. We will, however, look at some scripting examples and how you can gain direct access to various server data like the time and date.

The second section of the chapter will move from an overview of ASP, to using ASP with COM. ASP enables you to access the component technology that Microsoft provides for other areas of Windows. You can create ActiveX controls and use them in your ASP script. In fact, that's what we'll do in this section. We'll create an ATL ActiveX control, then display it using ASP.

Finally, we'll look at the combination of ISAPI, ASP, and COM+. This combination allows you to create very complex and flexible Web pages that can respond to any client need with a minimum of resources. We'll look at a simple example of all three of these technologies working together. You can use this simple example to create more complex examples of your own.

In This Chapter

Understanding how ASP can make Web-page development easier

Documenting the need for technology solutions using statistical measurements

Testing performance of an application using Web loading

Speeding up older technology solutions

Using ASP to access COM components

Making your Web pages easier to use with ASP and COM

Mixing ASP, ISAPI, and COM+ to create moderately complex Web pages

Determining when to use a mixed ASP and ISAPI environment

Creating a mixed-environment application

Understanding ASP

As previously stated, this chapter isn't going to provide a tutorial on using ASP. This section will provide an overview of ASP as it relates to more complex programming tasks that involve both COM and ISAPI. However, it's important to note that ASP files are text files with an ASP file extension. They can contain three kinds of data:

- **Text:** Essentially the data that the user is looking for. Every useful Web page contains some text that forms the usable data.
- **HTML Tags:** Any tag that you can use on a standard HTML page will also work just fine within an ASP file. Obviously, the browser must support the HTML tags that you want to use. For the most part, the HTML tags will allow you to format the page or the data it contains.
- **Script Statement:** You can include two kinds of script within an ASP file: client-side and server-side. We'll deal mainly with server-side scripting in this chapter.

Many ASP pages will start out as standard HTML pages. You'll set up an initial Web site and want to test the various links between pages. However, as the complexity of some pages grow, you'll find that you can no longer do what you need to do using a combination of text, HTML tags, and client-side scripts. You need to represent a server connection on the Web page in some way. That's when you need to consider using ASP, which allows you to work with server-side scripting and, most importantly, server-side objects.

You can use any scripting language with ASP that supports the ActiveX Scripting standard from Microsoft as long as you have the appropriate scripting engine. Microsoft ships two scripting engines with the default IIS implementation: JScript (Microsoft's version of the JavaScript language) and VBScript. According to Microsoft, you can get third-party scripting engines that support REXX, Perl, and Python. Other scripting engines may follow in the future.

An Overview of Server-side Scripting

If you've ever used JScript or VBScript to create a client side script within a Web page, then you already have the knowledge required to create the scripting content of an ASP file. You have to keep a few new rules in mind as you create your server-side ASP script.

The main benefit of using ASP is that you don't have to rely on the client to process the Web page. Unless you include a client-side script in the Web page itself, the server does all processing of the ASP script. The major drawback of using ASP is that the server can get loaded faster by the client due to the amount of processing that has to be performed. You need to take the extra processing load the ASP incurs into account when determining whether to use client-side processing, server-side processing, or a combination of both.

The first thing you need to know is that server-side scripts are always placed within two delimiters, <% and %>. When the server sees the <% delimiter, it knows that it's starting a server-side script and will execute the instructions that it finds. Likewise, when the server sees the %> delimiter, it knows that the script has finished and that any content should go to the client unchanged.

ASP scripts also support the notion of procedures and functions. A procedure does some internal processing, but doesn't return a value. The whole purpose of a procedure is to perform some work in the background. It might also display a dialog to the user. On the other hand, a function does return a value that gets displayed as part of the Web page. For example, a script might accept two input values, then return the summation of those values. You create a procedure when you use the <% delimiter by itself. Adding an equal sign like this, <%=, turns the procedure into a function. (The Microsoft documentation calls this an output directive, but the result is more like using a function than anything else — you can substitute the Response.Write( ) method for the output directive.)

The directory containing an ASP file must have the Script permission set as a minimum. Depending on what other files the directory contains, you may need to enable the Execute permission. You can use either the Script or Execute permission to enable ASP script-processing — you don't need to enable both permissions.

Inline scripts, those that create content on the fly while the server is reading the Web page, always use the default language supported by the Web server. By default, IIS assumes that you want to use VBScript as your primary scripting language. However, it's easy to change the default scripting language to something else by including a simple processing directive command like this:

```
<%@ LANGUAGE=JavaScript %>
```

Notice that the beginning script delimiter uses an @ (at) sign. Any time you see the @ sign as part of the beginning delimiter, you know that you're looking at a processing directive of some type. Directives can perform a wide variety of tasks, but they almost always affect the settings of the Web server. Table 10-1 contains descriptions of the most common processing directives.

Table 10-1
Common ASP Processing Directives

Directive	*Description*	*When to Use*
CODEPAGE	Determines the character encoding used for the Web page.	The choice of code page is normally determined by the language you want to use to display text on the Web page.

Continued

Table 10-1 *(continued)*

Directive	*Description*	*When to Use*
LCID	Determines the locality settings for the Web page.	Like the CODEPAGE keyword, this keyword is normally language or nationality specific. LCID affects the locality settings that affect things like how dates and monetary amounts get displayed.
TRANSACTION	Tells the Web server that the Web page will run under as a transaction.	Unlike the normal Web page where part of the process could fail and the user would still see a page, a transactional Web page is an all or nothing event. Using transactions for business purposes ensures that both client and server get everything needed to complete the current step. The most common use for transactions is in conjunction with databases. ASP transactions rely on Microsoft Transaction Server (MTS). We'll look at transactions in greater depth in Chapter 14.
ENABLESESSION STATE	Allows the Web server to create a single session for the entire time the user remains connected.	A session is a single instance of a connection between a client and a server. HTTP doesn't support sessions natively, which means that every time a user clicks a link, HTTP views it as an entirely new request. In other words, the server forgets who the user is, making it impossible to track what the user has done on the Web site. The inability of a standard Web server to remember what the user has done in the past makes it difficult to write e-commerce applications in which the application may need to track the items that the user has selected on multiple pages. This keyword allows you to create a session object that enables you to track the user's activities while he or she remains connected to the server. Obviously, this feature does incur a performance penalty, so you should use it with care.

Besides creating inline scripts using the <% and %> delimiters, you can create standard scripts that run at the server instead of the client. The secret here is to use the RUNAT argument and specify SERVER as the value when creating your script. For example, the following script will run at the server instead of at the client.

```
<SCRIPT LANGUAGE=JavaScript RUNAT=Server>
function Hello()
{
    Response.Write("Hello");
}
</SCRIPT>
```

One odd thing about using ASP that actually comes in pretty handy is that you can mix HTML, text, and script within the same block and actually expect the script to execute. About the only time you would use this feature is if you wanted to create a short script that relied on some type of conditional statement to determine what kind of output the user would see. Here's an example of how you might mix text, HTML, and ASP script together to produce two alternatives of the same Web page (depending on user preference in this case):

```
<% if (lNoGraphics) { %>
    You should be seeing a house.
<% } else { %>
    <IMG SRC=House.GIF>
<% } %>
```

In this case, if the user has chosen not to display graphics, he or she will see some text saying that he or she should see an image of a house. On the other hand, if the user did choose to display graphics, then we use the <IMG> tag to display the image of the house.

Notice that the same code includes the curly braces. You must include the curly braces if you're using JScript to create your script. Obviously, VBScript doesn't require curly braces, just as it doesn't when you write a client-side script.

ASP and Objects

ASP gives you full access to the capabilities of the scripting language that you use to create the application. In addition to the capabilities of the scripting language, you also gain access to built-in objects that make the job of working with the Web server much easier. The following list provides a description of the various objects that ASP provides access to as a default.

Application Allows you to share information between all of the users of a particular application.

Request	We talked about the HTTP request header in the "Understanding Request Headers" section of Chapter 9. This object allows you to access all of the information that the client provides as part of the request header, which can vary according to the type of request being made. The request-header information will include any data sent as part of a GET or POST request, cookies, client certificates, and any binary data provided as part of a file upload.
Response	Provides a means for sending information to the client from the server. You can use this object to do things like output text from a script. In addition to sending information for the user to see, you can also use the Response object to send a cookie or redirect the user to another URL.
Server	Allows you to access all of the methods and properties of the server. The Server object allows you to apply URL or HTTP encoding to strings, map a virtual path to a physical path, or set the timeout for executing a script. In addition, you'll use the CreateObject( ) method to create an instance of an ActiveX component (as we'll see later in this chapter).
Session	Stores the user settings for a particular session. A session lasts from the time the user makes an initial connection to the time that the user terminates the connection, or the session timeout is reached. Normally HTTP creates a new session every time a user moves from one page to the next. The number and type of variables stored in the Session object depends on the requirements of your application.
ObjectContext	Commits or aborts a transaction initiated by the ASP script.

Working with Collections

Some ASP objects support the concept of a collection. A collection is like an array, except that it gets stored within the instance of the object. In addition, the contents of a collection move around as you add or remove items from the list. Finally, unlike an array, each element of a collection has a name, which allows you to access the contents of the collection using either an index or an item name.

Collections are very handy to have around because you can use them to customize your code. For example, the Server object will allow you to store a collection of

strings that contain information about the current session. These strings could conceivably contain anything you want, like the user's name or the time of the last transaction. Because the collection is part of the object, the data for that object moves around with the object itself, making each object unique.

Using a collection is relatively easy. All you need to do to add an item to a collection, (like the one provided by the Server object) is add an item to the list using the appropriate method. Here's an example of how you would add items to the Server object collection:

```
<%
    Server.Contents("FirstItem") = "One";
    Server.Contents("SecondItem") = "Two";
    Server.Contents("ThirdItem") = "Three";
%>
```

There are two methods of accessing the values inside a collection. You can use an index for the item (collections are indexed starting at 1) or you can access the value by its name. For example, in the code below, both variables would contain a string with a value of two.

```
<%
    var cMyString1 = Server.Contents("SecondItem");
    var cMyString2 = Server.Contents(2);
%>
```

When Older Technology Is a Must

Your company may have been in the Internet business for quite some time when you read this chapter and the two before it. Sure, the capabilities provided by technologies like ISAPI, ASP, and COM sound like a dream come true, but you have a wealth of older technology like CGI code sitting around that the boss won't be happy about getting rid of. An existing investment in older technology can hold your company back and make it harder for you to show that the Internet is really the kind of place your company needs to invest in.

There are actually two answers to the question of making older technology work in today's environment. The first is the need to show that you really need something newer to deal with today's computing environment. The way to do that is to get quantifiable load statistics results from Web-server testing. After all, if you can't show that the new technology is actually faster, then you can't expect anyone to pay much attention to it.

Products like WebLoad from RadView Software, Inc. can make it easier to produce quantifiable input using a test system. WebLoad allows you to load your Web server with phantom users who simulate the actual kind of load that you'll see on your production server. Using

Continued

(continued)

an automated load test program like WebLoad allows you to show that the new technology is better able to handle today's e-commerce needs. In addition, you can use this kind of software to fine-tune any new ISAPI Filters, ISAPI Extensions, or ASP scripts that you create. You can find out more about WebLoad from the RadView Software site at `http://www.radview.com`.

It's not always possible to replace older technology with newer technology—at least not immediately. Your company may have a sizeable investment in those CGI applications, and even if the boss decided to replace those applications today, you don't have the resources to do it. Obviously, you'll want to keep that old CGI application around until you do have the time and resources to replace it.

Fortunately, you can improve the performance of existing technology solutions to your Internet needs in two ways. One of them is to conduct a survey of your Web site to look for potential performance killers like bad links and larger graphics files. If you've got the time to conduct a survey yourself, you can use a number of Web-analysis tools on the market that will do the job. However, what happens if you don't have the time or expertise to find all of those performance killers lurking on your Web site?

A new breed of company like Monte Sano Software is coming to the rescue of many developers. Their product, NetMechanic, can help you find all of those performance killers with little effort. All you need to do is go to their Web site at `http://www.netmechanic.com`, fill out a form, and then let Monte Sano Software do the rest. You'll get a confirmation of the Web-site monitoring they'll do to your site via e-mail. You'll be able to check Monte Sano's Web site for statistics about your Web site (all information is password-protected for confidentiality). There are actually two versions of this service. The first is free and allows you to perform a general one-time check of your Web site. The second service costs money and provides 24-hours-a-day, 7-days-a-week monitoring of your Web site.

Another solution to your performance problems may be making those CGI programs written in Perl or TCL/TK more efficient. Binary Evolution produces a product named VelociGen, a product that helps speed CGI applications up by allowing them to run in a separate process from the Web server. This separate session also supports script caching and allows the CGI script to maintain a connection to back-end databases. In addition to making your CGI application faster, running them in a separate process also makes the CGI application more reliable and less likely to crash the server. You can find out more about VelociGen at `http://www.binaryevolution.com`.

Using ASP with COM

At this point, you should have a pretty good overview of the capabilities of ASP. It's time to look at how ASP works with COM and ActiveX components. The following

sections will help you understand how ASP and COM can work together. In the first section, you'll learn how to use an ActiveX component within an ASP script. The focus of the next section is the ActiveX controls that come with ASP. Once you have an understanding of how ASP and COM can work together, we'll build an ATL-based ActiveX control to use within a script. Finally, we'll put the ActiveX control to work in a sample ASP application.

Using Objects within an ASP Script

The first question we need to answer about using ASP and ActiveX controls together is how you create a COM component within an ASP script in the first place. The first thing you need to understand about server-side component programming is that the component can't contain a user interface. The reason is simple: No one would be able to see the user interface or interact with it. In other words, a server-side component has to be self-contained and provide some type of output that the Web server can send to the client, usually in the form of text. This is an important difference between the ActiveX components you create for the client (like a pushbutton) and ones that you create for the server (like a sales-tax calculation).

There's nothing to stop you from including client-side ActiveX components that do have a user interface within your script. All you need to do is use the <OBJECT> tag. This HTML tag allows you to define a downloadable ActiveX control that will get registered on the user's machine. Once the control is registered, the Web page can display the component, just as if the component had been on the client system all along.

Once you have a component that will actually work on the server side, you need to create it within the script, just as you would within any application. Use the CreateObject( ) method of the Server object to create an instance of the component for JScript as shown here:

```
<%
    var oAd = Server.CreatObject("MSWC.AdRotator");
%>
```

Notice that we're not using the native JScript New( ) method for creating the object as you would when working with a client-side script. You must use the CreateObject( ) method of the Server object as shown to create an object that the ASP script can use. Otherwise, ASP won't be able to track the object. This could result in an inability of your script to access the object and will most likely create other problems like memory leaks within the application. (Memory leaks could eventually cause your server to crash when it runs out of memory; especially if your Web server receives heavy traffic.)

As an alternative to using the Server.CreateObject( ) method, you can always create a server-side component using the HTML <OBJECT> tag. Be sure to specify the RUNAT argument with a value of Server when you use this technique. Adding RUNAT=Server to the <OBJECT> tag declaration forces the component to get instantiated and run at the server rather than the client.

Many ActiveX components provide methods and properties. You access ActiveX control methods in ASP just like you would anywhere else. Follow the object name with a period and the name of the method you want to use. For example, if you created an instance of an object named oMyObject and the component has a method named DoIt, you'd access this method as oMyObject.DoIt( ).

Properties work much the same as well. All you need to do is use the same dot syntax you normally use to access the property. For example, oMyObject.MyProp = 0 would assign a value of 0 to the MyProp property in oMyObject.

The COM Connection

As previously mentioned, ASP, like most Microsoft products, allows you to use COM in the form of the ActiveX component technology. In fact, the most recent version of IIS comes with a few new specialty components that should make your life a lot easier when it comes time to create a new script. The following list will describe each of these new components and what they can do for you.

Database Access — The database access component makes it easy to access a database from the Web page. It allows you to create dynamic content for the user, just as if they were accessing the database through a network connection. In fact, the user has access to most database features, including the ability to manipulate tables and create queries.

Ad Rotator — Essentially this is an ActiveX version of the banners that you see at the top of many commercial Web pages on the Internet. You can load images and assign an URL to them. The user is automatically taken to the link site when he or she clicks the image. This component relies on a text data file containing a list of images and URLs to display.

Browser Capabilities — This component allows you to detect the current browser capabilities and tailor the Web-page content as appropriate for the browser in question.

File Access	In some cases, you need to access files on a user's machine. This component gives you full access to the user's file system. Obviously, you need to use this component with care because it would be fairly easy to damage the user's hard drive using it. Obviously, you'll need to disable any file system protection like Lockdown2000 on the user's machine before you'll be able to access the drive.
Content Linking	Think of this as a site map in text format. It allows you to create a list of sequential ASP file links in a single file rather than maintain those URL references in a number of ASP files.
Collaboration Data Objects	In short, this component provides you with mail capability. You can use it to allow users to send mail to each other. However, this component shouldn't be confused with a full-fledged e-mail reader. It really does help with collaboration in a small group and isn't designed for use as a complete e-mail system.
MyInfo, Status, System, and Tools	This component provides four essential tools for learning more about the client and server, which in turn can help you create tailored content for your Web site. The MyInfo component keeps track of personal information like the user's name. It also keeps track of Web site settings like whether the Web site has a guest book. The Status component enables you to determine some server statistics like the number of Web-site visitors and the number of HTTP connections. The System component is Macintosh specific — it enables a Macintosh system to create the Status object. The Tools component provides a lot of gizmos, including a random number generator. It also enables you to check for the existence of a file and process an HTML form.
Page Counter	A simple component that keeps track of the number of times that a page has been requested. It writes this number to disk on a regular basis so that the number won't be lost in the event of a server shutdown.
Permission Checker	You can use this component to determine whether a user has access to a given Web page or other resource. This enables you to customize the current page so that the user sees only the options he or she can actually access rather than all of the resources currently available on the server.

Creating an ATL-based ActiveX Control for Web Page Use

We're finally ready to create an ActiveX control that's designed for use with our server-side ASP script. This particular control will compute the temperature in either Celsius or Fahrenheit, depending on what the user provides as input. We'll use two methods; one will perform the conversion one way, whereas the other method will perform the conversion the other way. The following sections will tell you how to put this control together.

CD-ROM Location: Chapt10\TempConv
Program Name: TempConv.DLL
Source Code Modules in Text: TempConv.CPP

Creating the TempConv Control Shell

This section will show you how to create a simple ATL-based control for use within a server environment. This control won't include any form of user interface because you can't use controls of this type on the server. We'll take a closer look at ATL-based controls that provide a user interface in Chapters 11 and 12. The actual programming example appears in Chapter 12. The following steps will get you started with this ActiveX control example. (To follow this procedure, you have to have already started Visual C++ and be using version 6.0 or above.)

Try This

1. Use the File | New command to display the New dialog box shown in Figure 10-1. Notice that the ATL COM AppWizard entry is highlighted.

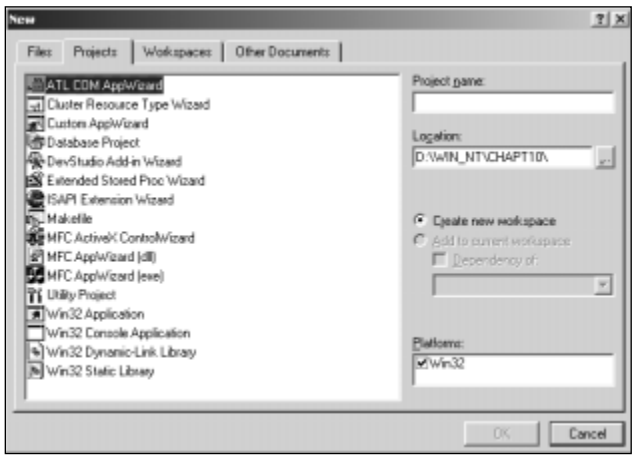

Figure 10-1: You'll need to choose the ATL COM AppWizard entry in the New dialog to start an ATL-based ActiveX control.

2. Type a name for your ActiveX control in the Project name field. The example uses TempConv as an application name, though you can use any name you'd like.
3. Click OK. You'll see the ATL COM AppWizard - Step 1 of 1 dialog box shown in Figure 10-2. This is where you'll choose the main file options like the kind of file created and the level of support that it provides. In this particular case, most of the default settings will work fine since we require neither MFC nor MTS support. In addition, ATL-based ActiveX controls normally start out as DLLs. We will want to merge the proxy stub code so that there aren't two DLL files to manage.

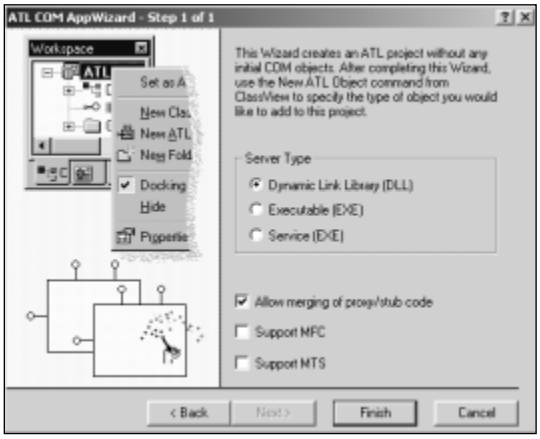
ATL COM AppWizard - Step 1 of 1
Workspace
Set as A
New Clas
New ATL
New Fold
Docking
Hide
Propertie
This Wizard creates an ATL project without any initial COM objects. After completing this Wizard, use the New ATL Object command from ClassView to specify the type of object you would like to add to this project.
Server Type
Dynamic Link Library (DLL)
Executable (EXE)
Service (EXE)
Allow merging of proxy/stub code
Support MFC
Support MTS
< Back
Next >
Finish
Cancel

Figure 10-2: The ATL COM AppWizard - Step 1 of 1 dialog box lets you to choose the main ActiveX control settings.

4. Check the Allow merging of proxy/stub code option. This will ensure that we end up with a single file that we'll later rename as an OCX file for use in the application.
5. Click Finish. You'll see the New Project Information dialog shown in Figure 10-3.
6. Click OK. Visual C++ will create the new project for you. This project won't include much; it won't even include any object code. We'll add an ATL object to the project in the next section.

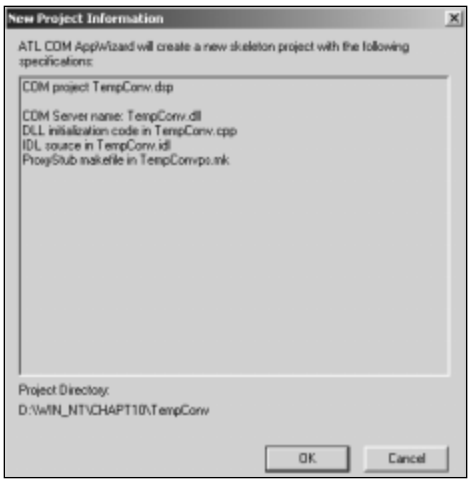

Figure 10-3: The New Project Information dialog shows which options you've chosen for the ActiveX control.

Inserting a New Object

The program shell we created in the previous section doesn't really do much. You could compile it, but the resulting DLL is pretty much wasted space. The reason is simple: There aren't any objects within the DLL for ASP (or anything else for that matter) to use. In this section of the chapter, I'll show you how to insert an object into the project. However, the object that we'll insert is a specialty object designed for use with ASP. It doesn't support an interface of any kind and provides only the interfaces that ASP will need. In other words, this ActiveX control really won't work in too many environments—it's specially tailored for use as a server-side scripting component. The following procedure will show you how to get the object inserted.

Try This

1. Use the Insert | New ATL Object... command to display the ATL Object Wizard dialog box shown in Figure 10-4. Notice that this dialog box categorizes the kinds of objects that you can insert by type, and several discrete object types are in each category. It's important to know what kind of object you want to insert into a project because not all ATL objects are created equal.
2. Highlight the ActiveX Server Component icon as shown in Figure 10-4, then click Next. You'll see the ATL Object Wizard Properties dialog shown in Figure 10-5. This is the dialog where you'll begin to define the object that you'll eventually use with ASP.

Figure 10-4: The ActiveX Server Component object provides special features used with ASP scripts.

Figure 10-5: The ATL Object Wizard Properties dialog lets you define many object properties, including the name as shown here.

3. Type **ConvertTemp** in the Short name field. Notice that the wizard automatically fills in the other fields for you. In most cases, these other values will work just fine. You should always look at them though to make sure that none of the values will conflict with something you already have in place.

The Attributes tab of the ATL Object Wizard Properties dialog contains important settings that determine how the object gets instantiated. For example, this tab determines what type of threading the object will use. We'll cover this tab completely in Chapter 12. The default settings will work just fine for this example, so we won't cover this tab here.

4. Click the ASP tab of the ATL Object Wizard Properties dialog. Figure 10-6 shows what you'll see. This tab contains specialty settings for this object type. As the book progresses, we'll discuss other types of specialty ATL object tabs. See the "ASP Object Settings" sidebar for details on the contents of this tab.

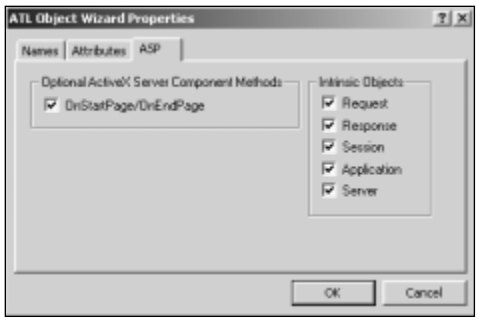

Figure 10-6: The ASP tab of the ATL Object Wizard Properties dialog allows you to choose which ASP objects and methods the new component will contain.

5. Ensure all of the options on the ASP tab are checked, then click OK. Visual C++ will add the required classes to your application. In addition to the class-specific items required to implement your component, Visual C++ will add code and interfaces required to implement specialty items like the ASP objects.

ASP Object Settings

The ActiveX Server Component object provides special settings that deal with the way in which the component will interact with ASP. You'll find these settings on the ASP tab of the ATL Object Wizard Properties dialog shown in Figure 10-6.

OnStartPage/OnEndPage is an optional checkbox that adds the OnStartPage and OnEndPage methods to your component. You'll need this option if you want to use ASP intrinsic objects. The intrinsic objects include the Request, Response, Session, Application, and Server objects described in the bullets that follow. Essentially, the OnStartPage( ) method gets called whenever a user requests an ASP Web page. The component uses the ScriptingContext class during that call to retrieve pointers to the standard objects that ASP supports. The object pointers get released during the OnEndPage( ) method call.

The Request, Response, Session, Application, and Server checkboxes enable access to these ASP objects. These objects perform the same tasks they do within an ASP script, so you can use them to do things like gain access to the server and make use of any collections that might be stored within the object. See the "ASP and Objects" section of this chapter for further details.

Adding Some Code to the ConvertTemp Class

We now have an application shell and an object that won't do anything, but at least we're getting closer to creating a functional end result. It's time to add some code to the ConvertTemp class. Obviously, that means adding some methods to it. The following procedure will tell you how to add methods to the object — this same procedure will work for any ASP object that you create.

Try This

1. Right-click the IConvertTemp entry in the ClassView tab of the Workspace window and choose Add Method... from the context menu. You'll see the Add Method to Interface dialog box shown in Figure 10-7.

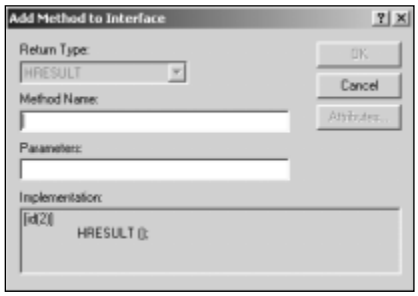

Figure 10-7: The Add Member Function dialog allows you to add new functions to your object.

2. Type **Fahrenheit2Celsius** in the Method Name field. This defines the name that you'll use to access the method from an ASP.
3. Type **[in] int iTemp, [out, retval] BSTR* bstrConvert** in the Parameters field. This defines the input and return values of the function. In this case, iTemp is an input variable that the user will supply, whereas bstrConvert is an output and return value that the method will supply.
4. Click Attributes and you'll see the Edit Attributes dialog shown in Figure 10-8. This dialog contains a list of method attributes — two of which are always defined as shown in the figure. The one attribute that you'll normally want to change is the helpstring. The default entry is based on the method name you type and won't be very helpful to the user.

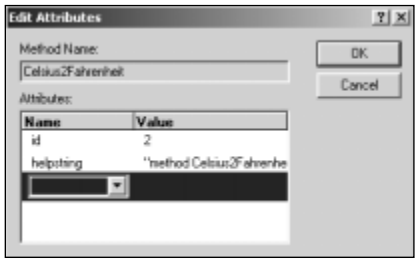

Figure 10-8: The Edit Attributes dialog allows you to change the attributes for a new method.

5. Type **"Converts Fahrenheit temperatures to Celsius."** in the helpstring field, and then click OK. Make sure you include the double quotes with the entry, or your control won't compile correctly.
6. Click OK. Visual C++ will add the new method to the IConvertTemp interface for you.
7. Repeat steps 1 through 6 to add a Celsius2Fahrenheit function to the CConvertTemp class. Give it the same input and return values.

Now that we have the method shells in place, it's time to add some code. Listing 10-1 shows the code you'll need to add for this example.

Listing 10-1: Fahrenheit2Celsius() and Celsius2Fahrenheit() Methods Source Code

```
STDMETHODIMP CConvertTemp::Fahrenheit2Celsius(int iTemp, BSTR *bstrConvert)
{
    LPTSTR      lpszValue = " ";    // Working value of iTemp.
    CComBSTR    bstrValue;          // Temporary BSTR holder.

    // Convert the input temperature to Celsius.  In addition,
    // convert the resulting value to a string.
    itoa(((iTemp - 32) / 9) * 5, lpszValue, 10);

    // Place the resulting value in a CComBSTR, then return
    // it to the user as a BSTR.
    bstrValue = lpszValue;
    *bstrConvert = bstrValue.Detach();

    // Return the standard value.
    return S_OK;
}

STDMETHODIMP CConvertTemp::Celsius2Fahrenheit(int iTemp, BSTR *bstrConvert)
{
    LPTSTR      lpszValue = " ";    // Working value of iTemp.
    CComBSTR    bstrValue;          // Temporary BSTR holder.

    // Convert the input temperature to Fahrenheit.  In addition,
    // convert the resulting value to a string.
    itoa(((iTemp * 9) / 5) + 32, lpszValue, 10);

    // Place the resulting value in a CComBSTR, then return
    // it to the user as a BSTR.
    bstrValue = lpszValue;
    *bstrConvert = bstrValue.Detach();
```

```
    // Return the standard value.
    return S_OK;
}
```

As you can see, the actual conversion code is relatively simple. In both methods, the input variable, iTemp, is converted to either Fahrenheit or Celsius. The result is converted to a string, then placed within a CComBSTR (this conversion ensures that the new temperature value will get converted from an LPTSTR to a BSTR, which is what the client requires). Finally, we place the value in the CComBSTR into the return value for the client.

At this point, you can compile the new component. What you'll end up with is a relatively small DLL. You don't need to rename the DLL as an OCX because we can register it on the server using the DLL file extension and the client will never see this file.

Registering TempConv on the Web Server

Simply placing the DLL we've just created on your server isn't going to do much because there isn't any way for ASP to know that it exists. That's why you need to register the DLL with the Windows registry. Registering the component tells clients where to find any required files and the component name.

Fortunately, registering a component is easy. All you need to do is open a command window using the Start | Command Prompt command, change directories to the location of the component file, then type **RegSvr32 TempConv.DLL** (or the name of any other component file). If the registration process is successful, you'll see a dialog like the one shown in Figure 10-9. Otherwise, you'll receive an extremely cryptic error message stating that the registration failed and the reason why.

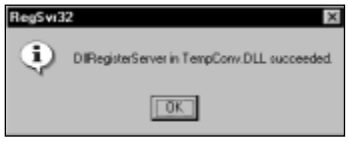

Figure 10-9: This is an example of a successful DLL registration.

If the registration does fail, normally the error code will give you some hint as to what happened. However, before you spend hours trying to figure out what that error code is all about, there are a couple of common problems that you can check. The most common problem is that the file is corrupt. Try placing a fresh copy of the file in the directory after you compile it again. Another possible problem is that you aren't in the same directory as the DLL. Make sure that RegSvr32 can actually find the file by performing a DIR command at the command prompt. You'll also

want to check your source code to make sure that all of the registry information is correct. Normally, the wizard does a good job of getting this information correct, but there are times where a registry key can be incorrect, or a missing quote will cause problems.

Obviously, if you use your machine to develop a lot of components, you won't want to clutter your registry with entries for all of them. Removing component entries is fairly automatic as long as you do it before you remove the component file. Just type **RegSvr32 -U <Component File Name>** at the command prompt and press enter. You'll get a success dialog just like you did when you registered the component.

Make absolutely certain that you unregister the component before you move it to another location, or delete it from the drive. Failure to do so will result in a registry that contains component references that no longer exist and could cause your machine to experience various kinds of problems like lockups.

Combining ASP and ActiveX

ASP and ActiveX can make good partners. Although ASP is a fairly complete scripting language, it doesn't always have the access to the server it requires. In addition, scripting languages are slow — you pay a performance penalty for using them. ActiveX represents one way to speed up the execution of a script.

CD-ROM Location: Chapt10\TempConvTest
Program Name: TempTest.ASP
Source Code Modules in Text: TempTest.ASP

The previous sections have shown you how to build an ATL-based ActiveX component specifically designed for server use without any kind of interface. Now that we have a component to use, let's view it in action. Listing 10-2 shows the source code for a very simple ASP file. Notice the combined used of text, HTML tags, and ASP script in the same file.

Listing 10-2: **TempTest Web Page Source Code**

```
<HTML>
<HEAD>
<TITLE>ATL Based ActiveX Control for ASP Test</TITLE>
</HEAD>

<BODY>

<!- Set the scripting language. ->
<%@ LANGUAGE=JavaScript %>
```

```
<!- Display a simple header. ->
<H1><CENTER>Temperature Conversion Example</CENTER></H1>

<!- Create the test object and data variable ->
<% var oTemp = Server.CreateObject("TempConv.ConvertTemp") %>

<!- Display the first temperature test message. ->
If the temperature is 212 degrees Fahrenheit, then it is:
<%= oTemp.Fahrenheit2Celsius(212) %>
degrees Celsius.<P>

<!- Display the second temperature test message. ->
Likewise, if the temperature is 100 degrees Celsius,
then it is:
<%= oTemp.Celsius2Fahrenheit(100) %>
degrees Fahrenheit.

</BODY>
</HTML>
```

As you can, this code is very simple. It begins with the usual combination of header, title, and body tags found on any Web page. Anyone who has designed a Web page will likely be familiar with these tags. Notice that I've included a heading of pure text and HTML tags along with the usual closing HTML tags. So from a Web-page design perspective, this file is very basic — to the point at which you couldn't make it much simpler without eliminating the content completely.

The first bit of ASP in this file is the directive that sets the language. In this case, we're using JavaScript because JavaScript should be easier for most C/C++ programmers to understand than VBScript. Remember that VBScript is the default ASP language, so you don't need to add a directive if you want to use VBScript to create your Web page.

Notice the first thing we need to do after defining the script language for the Web page is create an object variable. We discussed this method call earlier, but now you can see it in action.

Of course, the first thing on your mind should be the source of the object name in the listing. After all, we didn't create a component with that name. Actually, the default name for a server component is the filename that the component is stored in, a dot, and the name of the object within the file that you want to use. However, there's a quick method for verifying the name you should use for the CreateObject( ) method; just look in the registry with RegEdit. Use the Edit | Find command to locate the CLSID key entry for your component. Figure 10-10 shows the RegEdit display for our component. The VersionIndependentProgID key always contains the name you should use for the component, as shown in the figure.

Figure 10-10: RegEdit will allow you to find the name you should use to access your component within an ASP script.

As an alternative, you can always use the ProgID key value or the component's Globally Unique Identifier (GUID) if there's no VersionIndependentProgID key associated with the component. The GUID is the 128-bit number associated with the component. In the case of our example, the GUID is E8877300-A83E-11D2-BA9E-00A02456674B, which is hardly easy to remember, but very specific.

Now that there's an object variable to use, the code displays a test message and the results of using both of the calculation methods within the component. Notice that you only need to supply the input parameter; the [retval, out] parameter that we created in the ActiveX component is provided as an output of the method call. Figure 10-11 shows the output from our ASP script.

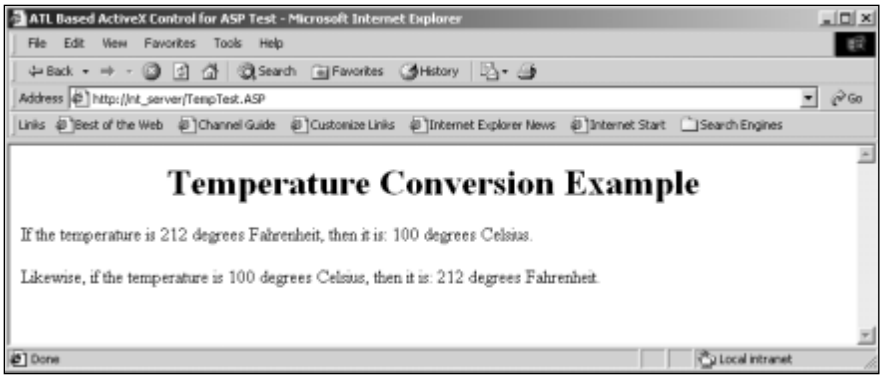
ATL Based ActiveX Control for ASP Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://nt_server/TempTest.ASP
Go
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start Search Engines
Temperature Conversion Example
If the temperature is 212 degrees Fahrenheit, then it is: 100 degrees Celsius.
Likewise, if the temperature is 100 degrees Celsius, then it is: 212 degrees Fahrenheit.
Done
Local intranet

Figure 10-11: The output from the TempTest ASP page shows the results of mixing ActiveX and ASP.

Using ASP with ISAPI and COM+

By now, you should have a better understanding of how ASP and ActiveX can work together. You can extend this capability to other technologies like ISAPI as well. In

fact, you can mix ActiveX, ASP, and ISAPI to create some really unique Web solutions that are both efficient and easy to create (or at least easier than working with one technology alone).

Let's consider just one part of the picture — ISAPI. In Chapters 8 and 9 we looked at ISAPI examples that created a complete solution. In other words, all of the ISAPI Extensions created a complete Web page and dealt with all of the complexities of working with multiple browsers. Some people view an ISAPI-only solution to Web-page development problems as too error-prone, too time-consuming, and too inflexible. Working with script code is much easier and definitely more productive from a programmer's perspective, at least when it comes to continually changing technology like browser capabilities. In addition, ASP is easy to learn, whereas working with ISAPI requires the services of an experienced C++ programmer. ISAPI Extensions also require more time to develop than any form of script. In other words, using ISAPI Extensions as a single solution is expensive and time-consuming from a human resources perspective.

Unfortunately, using script alone can be inefficient and slow — one of the major problems being a lack of capability. Scripts are loaded every time you execute them, which means the server must use extra processor cycles to read the hard drive. In addition, scripts are interpreted, which uses still more processor cycles during the execution phase of the Web-page generation cycle. Scripts also require more room in memory to perform the same services that a compiled application can perform. You also don't have access to advanced programming technologies like threads. The use of these advanced technologies can greatly optimize the meager resources that the server has to offer. Using ISAPI to actually generate the Web page is far more efficient than ASP (or any scripting language) could ever hope to be.

There is a middle-ground approach. What if you develop an ISAPI solution that creates only a specific kind of page given a certain set of requirements? ASP can determine which ISAPI Extension to call based on the capabilities of the browser making the request. Unfortunately, ASP by itself doesn't have the means to fully determine browser capabilities. That's why you'd combine the ASP script with the Browser Capabilities component that is part of IIS. In other words, you'd use three different technologies to create an easy-to-maintain, yet extremely efficient solution to the problem of supporting multiple browsers. That's the example we're going to visit in this section of the chapter — a solution to the multiple browser problems that's extremely efficient from both a programmer and hardware resources perspective. Using this technique will allow you to develop Web pages that execute quickly, require fewer resources on the server, yet also optimize development time.

In this example, we'll create two ISAPI Extensions. The first will work with browsers that can handle frames, whereas the second will work with browsers that can't. The ASP page will contain code that will load the Browser Capabilities component, determine which frames capabilities the browser has, then call the appropriate

ISAPI Extension to display the Web page. This solution may sound complex, but you'll find that it's actually quite easy to implement because the various components are compartmentalized. In other words, this is an example of modular programming revisited.

Creating the NoFrames ISAPI Extension

The first ISAPI Extension that we'll create for this example will look very much like the ones we've created in previous chapters, but with a little added functionality. In this case, we have to duplicate the features that we'll create for our framed version. The framed version will contain four frames as defined in the list below.

Upper-left Frame	The company logo
Upper-right Frame	A company name and page title
Lower-left Frame	A list of links to other places on the Web site
Lower-right Frame	The main content for the Web page

What we'll do to implement these features in this example is create a Web page with a logo, page title, and company name at the top. Obviously, this information will disappear as the user scrolls down the page, but at least the user will see it when he or she first visits the Web site. We'll then place a link to an anchor at the bottom of the page to the section of the page that displays the links that would normally appear in the lower-left frame. That way the user can go right to the list of links to other Web pages, should he or she need to drill down to a specific page instead of starting at the beginning of the Web site. After the link, you'll see the main content for the Web page. There won't be much main content because the page is only for demonstration purposes. Finally, we'll create a section containing the links to other pages on the Web site. Again, the example contains a couple of links to nonexistent Web sites for demonstration purposes.

CD-ROM Location: Chapt10\NoFrames
Program Name: NoFrames.DLL
Source Code Modules in Text: NoFrames.CPP

Creating the NoFrames ISAPI Extension Shell

We've created quite a few ISAPI Extensions in the book so far, so this section will contain a shortened version of the usual application shell creation procedure. If you want to see the full procedure, go to the "Writing a Simple ISAPI Extension" section of Chapter 8. Create a new ISAPI Extension project named NoFrames. The following procedure will show you an alternative method for creating the ISAPI Extension function for this example.

Try This

1. Right-click the CNoFramesExtension entry on the ClassView tab of the Workspace window, then choose Add Member Function... from the context menu. You'll see the Add Member Function dialog box shown in Figure 10-12. The Function Type field is where you provide the return type of the function. The Function Name field contains both the function name and any input parameters. This dialog box allows you to choose the level of external access to the function and both the static and virtual function modifiers.

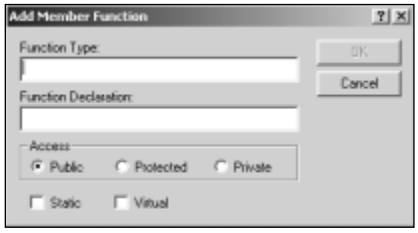

Figure 10-12: The Add Member Function dialog box allows you to create new functions with less work.

2. Type **void** in the Function Type field.
3. Type **DisplayPage(CHttpServerContext* pCtxt)** in the Function Name field.
4. Click OK. Visual C++ will add the new member function for you, including the proper header entries.

Adding Some Code to NoFrames

At this time our ISAPI shell is ready to go. We'll still need to add a parse map to the ISAPI extension. Because our function doesn't require any parameters, the parse map will consist of a single line of code. Listing 10-3 shows the code you'll need to add in bold.

Listing 10-3: **NoFrames ISAPI Extension Parse Map**

```
BEGIN_PARSE_MAP(CNoFramesExtension, CHttpServer)
   // Display a Web page that doesn't include frames.
   ON_PARSE_COMMAND(DisplayPage, CNoFramesExtension, ITS_EMPTY)

   // Perform the default action.
   ON_PARSE_COMMAND(Default, CNoFramesExtension, ITS_EMPTY)
   DEFAULT_PARSE_COMMAND(Default, CNoFramesExtension)
END_PARSE_MAP(CNoFramesExtension)
```

The Web-page code for this example will be a little more complex than previous examples because we're trying to duplicate the features of a Web page that uses frames, but the actual ISAPI Extension code is fairly simple. A production page would probably be a little more complex than the one shown here. This one is designed to show the techniques involved, not replicate production-level content. Listing 10-4 shows the code we'll use for the Web page without frames.

Listing 10-4: **DisplayPage Method Source Code**

```
void CNoFramesExtension::DisplayPage(CHttpServerContext *pCtxt)
{
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Create a table to display the company logo, company name, and
    // a tag line, and the page title.  This example will also include
    // comments that can be displayed with the Web page source.
    *pCtxt << _T("\r\n\r\n<!-- Create a table for displaying the header-->");
    *pCtxt << _T("\r\n<TABLE BORDER=0 ALIGN=CENTER WIDTH=90%>\r\n");
    *pCtxt << _T("<TR>\r\n");
    *pCtxt << _T("\r\n\t<!-- Display the company logo. -->\r\n");
    *pCtxt << _T("\t<TD VALIGN=CENTER WIDTH=200>");
    *pCtxt << _T("<IMG SRC=../Files/ABCLogo.GIF></TD>\r\n");
    *pCtxt << _T("\r\n\t<!-- Display the company name and tagline. -->\r\n");
    *pCtxt << _T("\t<!-- Also display the page name. -->\r\n");
    *pCtxt << _T("\t<TD VALIGN=CENTER><CENTER><H1>ABC Company</H1>\r\n");
    *pCtxt << _T("\t\t<EM>We Do Something!</EM>\r\n");
    *pCtxt << _T("\t\t<H2>Welcome to Our Home Page</H3></CENTER></TD>\r\n");
    *pCtxt << _T("</TR>\r\n");
    *pCtxt << _T("</TABLE>\r\n");

    // Display a link to the other links section of the Web page.
    *pCtxt << _T("\r\n<!-- Link to other links on Web page. -->\r\n");
    *pCtxt << _T("<P><EM>Click <A HREF='#OtherLink'>here</A> to see links\r\n");
    *pCtxt << _T("to other company sponsored Web sites.</EM>\r\n");

    // Display the main page content.
    *pCtxt << _T("\r\n<!-- Display the content for this page. -->\r\n");
    *pCtxt << _T("<H3><CENTER>ABC Company Events</CENTER></H3>\r\n");
    *pCtxt << _T("<STRONG>Strange But True: </STRONG>\r\n");
    *pCtxt << _T("The president learned today that his hair wasn't\r\n");
    *pCtxt << _T("on straight.  The chairman of the board and most\r\n");
    *pCtxt << _T("stockholders were outraged to learn of this bizarre\r\n");
    *pCtxt << _T("event.  To learn more about the implications of this\r\n");
    *pCtxt << _T("unreal hair revelation, click <A HREF='MyHair.HTM'>\r\n");
    *pCtxt << _T("here</A>.<P>\r\n");
```

```
    *pCtxt << _T("<STRONG>New Technology News: </STRONG>\r\n");
    *pCtxt << _T("Company scientists have made an astounding revelation\r\n");
    *pCtxt << _T("about a revolutionary new device called a door that\r\n");
    *pCtxt << _T("reputedly keeps flies out of the building.  Said door\r\n");
    *pCtxt << _T("includes appendages that allow people to enter and
leave\r\n");
    *pCtxt << _T("the building.  The scientists are also studying a
device\r\n");
    *pCtxt << _T("for windows called a screen.  To learn more about these\r\n");
    *pCtxt << _T("incredible inventions, click <A HREF='ADoor.HTM'>here\r\n");
    *pCtxt << _T("</A>.<P>\r\n");

    // Display the other links.
    *pCtxt << _T("\r\n<!-- Display a list of other links. -->\r\n");
    *pCtxt << _T("<HR ALIGN=CENTER SIZE=3 WIDTH=80%>\r\n");
    *pCtxt << _T("<A NAME='OtherLink'><CENTER><H3>");
    *pCtxt << _T("Links to Other Web Sites</H3></CENTER></A>\r\n");
    *pCtxt << _T("\r\n<UL>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.myplace.com'>My Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.yourplace.com'>Your Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.ourplace.com'>Our Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.theirplace.com'>Their Place</A>\r\n");
    *pCtxt << _T("</UL>\r\n");

    EndContent(pCtxt);
}
```

As you can see, there's nothing too spectacular in this code. All it does is implement the same kind of Web page that we'll implement using frames in the section that follows. If you executed this ISAPI Extension, you'd see a Web page similar to the one shown in Figure 10-13.

Creating the Frames ISAPI Extension

This section will tell you how to create a frame-based Web page using an ISAPI Extension. We'll replicate the Web-page conditions shown in the NoFrames ISAPI Extension coding example in the previous section.

CD-ROM Location: Chapt10\Frames
Program Name: Frames.DLL
Source Code Modules in Text: Frames.CPP

http://nt_server/controls/noframes.dll?displaypage? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://nt_server/controls/noframes.dll?displaypage?
Go
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start Search Engines
ABC
ABC Company
We Do Something!
Welcome to Our Home Page
Click here to see links to other company sponsored Web sites.
ABC Company Events
Strange But True: The president learned today that his hair wasn't on straight. The chairman of the board and most stockholders were outraged to learn of this bizarre event. To learn more about the implications of this unreal hair revelation, click here.
New Technology News: Company scientists have made an astounding revelation about a revolutionary new device called a door that reputedly keeps flies out of the building. Said door includes appendages that allow people to enter and leave the building. The scientists are also studying a device for windows called a screen. To learn more about these incredible inventions, click here.
Links to Other Web Sites
My Place
Your Place
Our Place
Their Place
Local intranet

Figure 10-13: The NoFrames Web page is designed to support browsers that don't allow the use of frames.

Creating the application shell is similar to the procedure for the NoFrames ISAPI Extension. In this case, the example uses a project name of frames. You can also use the same method for adding member functions to the ISAPI Extension that we used in the previous example. However, in this case, we need to create the five following methods:

- **MainFrm:** Contains the code for the main Web-page window. This frame page holds the four display windows.
- **IconWnd:** Displays the company icon. This is the upper-left window of the four-window frame.
- **TitleWnd**: Displays the company name, tag line, and the title for the Web page. This is the upper-right window of the four-window frame.
- **LinkWnd:** Displays the list of links to other Web sites that the company supports. This is the lower-left window of the four-window frame.

- **HomeWnd:** Displays the homepage that the user sees when first entering the company Web site. This window contains the main content for the Web page and will change for every Web page displayed. This is the lower-right window of the four-window frame.

There's an important consideration for this ISAPI Extension that we haven't talked about yet. The four display windows are standard HTML pages that implement a <BODY> tag pair. On the other hand, the frame page uses a <FRAMESET> tag pair, so it won't work if we use the standard WriteTitle( ) and EndContent( ) methods. What we need to do is define alternative WriteTitle2( ) and EndContent2( ) methods. You can use the same technique to add these two methods that you used to create the five page-specific methods. Remember that both methods will have a return value of void and use the same CHttpServerContext input parameter as the other methods.

We'll need a parse map for this ISAPI Extension. As with the previous example, we won't need to pass parameters to the member functions. However, you could easily pass a parameter to control the size of the frames, individual window features, and other adjustments to the Web-page display. Listing 10-5 shows, in bold, the parse map entries that you'll need to make.

Listing 10-5: **Frames ISAPI Extension Parse Map**

```
BEGIN_PARSE_MAP(CFramesExtension, CHttpServer)
    // Main Web page frame.
    ON_PARSE_COMMAND(MainFrm, CFramesExtension, ITS_EMPTY)

    // Icon window.
    ON_PARSE_COMMAND(IconWnd, CFramesExtension, ITS_EMPTY)

    // Company name, tag line, and page title.
    ON_PARSE_COMMAND(TitleWnd, CFramesExtension, ITS_EMPTY)

    // Links to other company sponsored Web sites.
    ON_PARSE_COMMAND(LinkWnd, CFramesExtension, ITS_EMPTY)

    // Main home Web page content.
    ON_PARSE_COMMAND(HomeWnd, CFramesExtension, ITS_EMPTY)

    // Perform the default action.
    ON_PARSE_COMMAND(Default, CFramesExtension, ITS_EMPTY)
    DEFAULT_PARSE_COMMAND(Default, CFramesExtension)
END_PARSE_MAP(CFramesExtension)
```

Once you have an application shell in place, along with the five member functions, you'll need to add some code. Listing 10-6 contains the code that you'll need for this part of the example. Notice that using frames makes the code more modular and easier to handle, it would be especially so for a Web page that's more complex than the one in our example.

Listing 10-6: Source Code for the Five Frame-based Web-page Methods

```
void CFramesExtension::MainFrm(CHttpServerContext *pCtxt)
{
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle2(pCtxt);

    // Create the four display windows.
    *pCtxt << _T("\r\n<!-Add an icon->\r\n");
    *pCtxt << _T("<FRAME SCROLLING=NO SRC=Frames.DLL?IconWnd?>\r\n");

    *pCtxt << _T("\r\n<!-Create a heading.->\r\n");
    *pCtxt << _T("<FRAME SCROLLING=NO SRC=Frames.DLL?TitleWnd?>\r\n");

    *pCtxt << _T("\r\n<!-List other sites that may be of interest.->\r\n");
    *pCtxt << _T("<FRAME SRC=Frames.DLL?LinkWnd?>\r\n");

    *pCtxt << _T("\r\n<!-Display the main content for our page.->\r\n");
    *pCtxt << _T("<FRAME SRC=Frames.DLL?HomeWnd?>\r\n");

    // End the Web page.
    EndContent2(pCtxt);
}

void CFramesExtension::IconWnd(CHttpServerContext *pCtxt)
{
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Display a logo in the upper left corner.
    *pCtxt << _T("\r\n<!-- Display the company logo. -->\r\n");
    *pCtxt << _T("<IMG SRC=../Files/ABCLogo.GIF>\r\n");

    // End the Web page.
    EndContent(pCtxt);
}

void CFramesExtension::TitleWnd(CHttpServerContext *pCtxt)
{
```

```
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Display the company name, tag line, and page title.
    *pCtxt << _T("\r\n<!-- Display the company name and tagline. -->\r\n");
    *pCtxt << _T("<!-- Also display the page name. -->\r\n");
    *pCtxt << _T("<CENTER><H1>ABC Company</H1>\r\n");
    *pCtxt << _T("<EM>We Do Something!</EM>\r\n");
    *pCtxt << _T("<H2>Welcome to Our Home Page</H3></CENTER>\r\n");

     // End the Web page.
    EndContent(pCtxt);
}

void CFramesExtension::LinkWnd(CHttpServerContext *pCtxt)
{
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Display the other links.
    *pCtxt << _T("\r\n<!-- Display a list of other links. -->\r\n");
    *pCtxt << _T("<CENTER><H3>Links to Other Web Sites</H3></CENTER>\r\n");
    *pCtxt << _T("\r\n<UL>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.myplace.com'>My Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.yourplace.com'>Your Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.ourplace.com'>Our Place</A>\r\n");
    *pCtxt << _T("\t<LI><A HREF='www.theirplace.com'>Their Place</A>\r\n");
    *pCtxt << _T("</UL>\r\n");

    // End the Web page.
    EndContent(pCtxt);
}

void CFramesExtension::HomeWnd(CHttpServerContext *pCtxt)
{
    // Start the Web page and display the default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Display the main page content.
    *pCtxt << _T("\r\n<!-- Display the content for this page. -->\r\n");
    *pCtxt << _T("<H3><CENTER>ABC Company Events</CENTER></H3>\r\n");
    *pCtxt << _T("<STRONG>Strange But True: </STRONG>\r\n");
    *pCtxt << _T("The president learned today that his hair wasn't\r\n");
    *pCtxt << _T("on straight.  The chairman of the board and most\r\n");
    *pCtxt << _T("stockholders were outraged to learn of this bizarre\r\n");
    *pCtxt << _T("event.  To learn more about the implications of this\r\n");
```

Continued

Listing 10-6: *(continued)*

```
    *pCtxt << _T("unreal hair revelation, click <A HREF='MyHair.HTM'>\r\n");
    *pCtxt << _T("here</A>.<P>\r\n");

    *pCtxt << _T("<STRONG>New Technology News: </STRONG>\r\n");
    *pCtxt << _T("Company scientists have made an astounding revelation\r\n");
    *pCtxt << _T("about a revolutionary new device called a door that\r\n");
    *pCtxt << _T("reputedly keeps flies out of the building.  Said door\r\n");
    *pCtxt << _T("includes appendages that allow people to enter and
leave\r\n");
    *pCtxt << _T("the building.  The scientists are also studying a
device\r\n");
    *pCtxt << _T("for windows called a screen.  To learn more about these\r\n");
    *pCtxt << _T("incredible inventions, click <A HREF='ADoor.HTM'>here\r\n");
    *pCtxt << _T("</A>.<P>\r\n");

    // End the Web page.
    EndContent(pCtxt);
}

void CFramesExtension::WriteTitle2(CHttpServerContext *pCtxt)
{
    // Create a standard heading.
    *pCtxt << _T("<HTML>\r\n<HEAD>\r\n");

    // Display a title
    *pCtxt << _T("<TITLE>Creating a Page with Frames</TITLE>\r\n");

    // Start the frame.
    *pCtxt << _T("\r\n<!-Set the frame parameters.->\r\n");
    *pCtxt << _T("<FRAMESET COLS='220, *' ROWS='220, *'>\r\n");
}

void CFramesExtension::EndContent2(CHttpServerContext *pCtxt)
{
    // End the frame.
    *pCtxt << _T("\r\n<!-- End the frameset. -->\r\n");
    *pCtxt << _T("</FRAMESET>\r\n");

    // End the Web page.
    *pCtxt << _T("\r\n</HTML>");
}
```

Almost all the HTML code for the example is the same as the previous one, except for the <FRAMESET> specific code in the MainFrm( ) method, so we'll look at the MainFrm( ) method first. The first change you'll notice is that we use the WriteTitle2( ) method in place of the more familiar WriteTitle( ) method. The reason is simple: We need to create a frame instead of the more familiar <BODY> tag. In fact, if you look at the WriteTitle2( ) method, you'll see very little difference

between it and the WriteTitle( ) method replacement we wrote for Chapter 8 except for the use of the <FRAMESET> tag instead of the <BODY> tag.

Normally frames use a SRC attribute that consists of references to other Web pages. Notice how the <FRAME> tags in this example differ. They call other methods within the current DLL to display the Web pages within the frame. One of the benefits of this approach is that everything needed to display the Web page appears in a single file instead of the five files that would have been required for this example. Not only is the page harder to break, but it uses disk space more efficiently as well because fewer clusters are only partially filled.

We haven't discussed a replacement for the EndContent( ) method before. This example shows that you can easily replace this method if necessary to produce a specific result. In this case, we need an end for a <FRAMESET> instead of a <BODY> tag, which is why the example uses the EndContent2( ) method. As before, there isn't anything special about the EndContent2( ) method other than the use of the </FRAMESET> tag.

The four Web-page display-page methods, IconWnd( ), TitleWnd( ), LinkWnd( ), and HomeWnd( ) all work like the standard ISAPI Extension methods that we've discussed in previous chapters. Each one of these methods displays a particular part of the frame Web page shown in Figure 10-14.

http://nt_server/controls/Frames.dll?MainFrm? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://nt_server/controls/Frames.dll?MainFrm?
Go
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start Search Engines
ABC
ABC Company
We Do Something!
Welcome to Our Home Page
Links to Other Web Sites
My Place
Your Place
Our Place
Their Place
ABC Company Events
Strange But True: The president learned today that his hair wasn't on straight. The chairman of the board and most stockholders were outraged to learn of this bizarre event. To learn more about the implications of this unreal hair revelation, click here.
New Technology News: Company scientists have made an astounding revelation about a revolutionary new device called a door that reputedly keeps flies out of the building. Said door includes appendages that allow people to enter and leave the building. The scientists are also studying a device for windows called a screen. To learn more about these incredible inventions, click here.
Local intranet

Figure 10-14: The Web page produced by the Frames ISAPI Extension is created by a single file rather than the five usually required by a page of this sort.

Designing the BrowseIt ASP Page

At this point, we have two different ISAPI Extensions that we can use to display content for the user. Obviously, the problem is finding a way to differentiate between the two so that the user gets the best possible content that the browser will support. That's what we'll do in this section: We'll create an ASP script that will decide which ISAPI Extension to call based on the capabilities of the client.

CD-ROM Location: Chapt10\BrowseCap
Program Name: BrowseCap.ASP
Source Code Modules in Text: BrowseCap.ASP

ASP doesn't provide a method for determining the browser capability by itself. In addition to the script, we'll also need to use an ActiveX component that can detect the browser's capability and provide input to the script. Fortunately, that component is provided to you as part of IIS in the form of the Browser Capabilities component. Listing 10-7 shows the source code you'll need to detect the type of browser being used, then select the proper ISAPI Extension.

Listing 10-7: **BrowseCap ASP Script Source**

```
<HTML>
<HEAD>
<TITLE>Automatic Selection of an ISAPI Extension</TITLE>
</HEAD>

<BODY>

<!- Set the scripting language. ->
<%@ LANGUAGE=JavaScript %>

<!- Load the ActiveX control. ->
<% var oBC = Server.CreateObject("MSWC.BrowserType") %>

<!- Decide which ISAPI Extension to select. ->
<% if (oBC.frames == true)
   {
%>

<!- If the browser does support frames, then create
    a script for changing the window's HREF to the
    ISAPI Extension for browsers the support frames. ->
<SCRIPT LANGUAGE=JavaScript>
      window.location.href = 'Controls/Frames.DLL?MainFrm?';
</SCRIPT>

<!- Otherwise, it must be an older browser that doesn't
```

```
    support frames. ->
<%
   }
   else
   {
%>

<!- So, we need a script that will load the non-framed
    ISAPI Extension. ->
<SCRIPT LANGUAGE=JavaScript>
      window.location.href = 'Controls/NoFrames.DLL?DisplayPage?';
</SCRIPT>

<%
   }
%>

</BODY>
</HTML>
```

You should notice a few things about this code. The most important of which is that we have discrete JavaScript and ASP script sections throughout the file. This example shows that you can mix script languages to good effect. In this case, we use a small, but very efficient script to detect a browser type, then call on an ISAPI Extension to display the correct page type.

As in the previous example, this example begins with a declaration stating that we'll use JavaScript as the ASP script language. The next thing the code does is create an instance of the Browser Capabilities component. We don't need to search the registry this time since the Microsoft documentation is very clear about the calling syntax for creating the component. Finally, we use the frames property of the Browser Capabilities component to determine which ISAPI Extension to call in simple if statements. Notice that the JavaScript sections merely change the window's location.

The Browser Capabilities component relies on the BrowsCap.INI file for property settings. The results you get are based on the contents of the HTTP User Agent setting passed to the server by the client. Though the BrowsCap.INI file does contain settings for most of the common browsers on the market, you may find that some browsers aren't supported. As a result, your ASP script may fail to detect certain classes of browse and provide the appropriate results to the client. Modifying the BrowsCap.INI file allows you to enhance the ability of the Browser Capabilities component to detect these other browser types. Obviously, you'll need to know enough about the unsupported browser to make the appropriate entries in the BrowsCap.INI file.

Summary

This is the final chapter on new Web technologies that you can access using the built-in capabilities of Windows 2000 and Visual C++. All of these technologies help you provide better content for the user that stresses the server less and improves system reliability. Obviously, some of these advantages come at the cost of increased application development complexity and time for delivery. Here are some additional conclusions you can draw from the contents of this chapter:

- ASP files contain three elements: text, HTML tags, and script statements.
- There are two types of script statement in an ASP file: server-side and client-side, both of which rely on standard scripting languages like JScript and VBScript.
- An ASP script resided between the <% and %> delimiters.
- Adding an equal sign to the beginning delimiter of an ASP script like this, <%=, tells the Web server that you want to output the result of the script to the client as content. It's the same as using the Response. Write( ) method.
- A processing directive begins with the <%@ delimiter. Processing directives allow you to change server settings like the default script language and the code page used to create text for the resulting Web page.
- Microsoft provides a set of default ASP objects that allow your scripts to better interact with both the client and server. These default objects include Application, Request, Response, Server, Session, and ObjectContext.
- Collections work similar to arrays. They're actually part of some ASP objects, which means that any data you store moves around with the object in question. A collection is always indexed beginning with 1, and you can access individual items using the item name rather than an index value if so desired.
- Load testing of your application is an absolute must. Not only does load testing show that the application will perform as expected once users actually start using it, but you can use the statistical information derived from the test to show the need for additional new technology tools.
- Older technology solutions need not be a dead end. You can usually improve application performance by adding new technology to the existing old technology infrastructure.
- Speeding the performance of old technology solutions is often a better solution than starting from scratch. You can find products on the market that help you look for performance problems as well as make the old technology application itself run faster.
- Creating most ASP-specific objects involves using the Sever.CreateObject( ) method in place of the standard object creation method provided by the scripting language.

- Microsoft has provided a wealth of new components with the latest ASP implementation. All of them are standard ActiveX components that are optimized for Internet use.
- Creating an ATL COM AppWizard project generates a simple empty shell. You need to add objects to the project before the resulting DLL can perform any work.
- Visual C++ provides a wealth of ATL object types to choose from. It's important that you choose the right object so that the wizard can create the appropriate interfaces for you automatically.
- One of the advantages to using ISAPI Extensions for creating a Web page that uses frames is that there are fewer files to deal with. All of the Web-page code appears in a single file.
- There are situations in which an ISAPI Extension may need more than one way to create beginning and ending tags for a Web page. In this case, you can create new methods to replace the more common StartContent( ), WriteTitle( ), and EndContent( ) method calls.
- Always remember to keep ASP and other script-language sections separate within the source file. Otherwise, you may find that code that should work won't because the ASP interpreter is confused.

✦ ✦ ✦